UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Agricultura de Barcelona

MONITORIZACIÓN DE CULTIVOS MEDIANTE NANOSATÉLITES CON CAPACIDAD DE COMUNICACIÓN BIDIRECCIONAL

Trabajo final del Máster

Master on Key Enabling Technologies 4 Food and + Bioprocess

Autor: Stephen Andreu Rhoton Meléndez
Tutores internos: Rafael Vidal Ferré
Marcos Quílez Figuerola
Tutor externo: David Paredes
Fecha: 2 / 7 / 2020

Resumen

El objetivo principal de este proyecto es validar los nanosatélites como tecnología de monitorización de cultivos. Para validar esta tecnología, se pretende diseñar y construir un prototipo preliminar que permita realizar una prueba de concepto, recogiendo datos de u ncultivo y procesarlos para ser transmitidos a un nanosatélite.

Para contextualizar la prueba, se realizó un estudio de cultivos de interés y parámetros a medir. Tras contrastar datos y considerar varios cultivos de alta producción, se escogieron la soja y el tomate. Ambos se producen en grandes cantidades en diferentes países del hemisferio norte y sur, el cual es un dato de interés para aprovechar los nanosatélites la mayor parte del año. Respecto a los parámetros, se escogieron la temperatura y humedad.

La prueba de concepto consiste en construir y poner en marcha un prototipo con tres sensores capaz de transmitir las medidas a un nanosatélite. Se ha escogido un sensor para medir la temperatura del suelo (THERM200), otro para medir la humedad del suelo (VH400) y un tercer sensor (DHT22) que mide tanto la temperatura como la humedad del aire. Los tres sensores se conectan a un microcontrolador (Arduino Uno), que se encarga de procesar las medidas para que sean transmitidas. Las medidas se envían a los nanosatélites mediante un terminal utilizando el protocolo KISS (del inglés *Keep It Simple, Stupid!*).

Se ha desarrollado un prototipo capaz de recoger las medidas de los sensores y almacenarlas en tramas KISS con el menor tamaño posible. Debido a circunstancias, excepcionales, no se ha podido tener acceso a los nanosatélites, de forma que la comunicación se ha comprobado entre dos nodos "terrestres". Queda pendiente verificar la comunicación con los nanosatélites. El prototipo está listo, es funcional y se deja preparado para cuando el acceso a los nanosatélites esté disponible. A pesar de que falta evaluar la comunicación con los nanosatélites, que es una parte fundamental del estudio, el trabajo de selección de cultivos, construcción del prototipo, recogida de datos y transmisión de la información se ha realizado satisfactoriamente.

Resum

L'objectiu principal del projecte és validar els nanosatèl·lits com a tecnologia de monitoratge de cultius. Per a validar aquesta tecnologia, es pretén dissenyar i construir un preliminar que permeti realitzar una prova de concepte, recollint les dades d'un cultiu i processar-los per ser transmesos a un nanosatèl·lit.

Per contextualitzar la prova, es va realitzar un estudi de cultius de interès i paràmetres a mesurar. Després de contrastar dades i considerar diversos cultius d'alta producció, es van escollir la soja i el tomàquet. Tots dos es produeixen en quantitats grans i en diferents països de l'hemisferi nord i sud, una dada interessant de cara a aprofitar els nanosatèl·lits la major part de l'any. Respecte als paràmetres, es van escollir la temperatura y la humitat.

La prova de concepte consisteix en construir un prototip amb tres sensors capaç de transmetre les mesures a un nanosatèl·lit. Es van escollir un sensor per mesurar la temperatura del sòl (THERM200), un altre per mesurar la humitat del sòl (VH400) i un tercer sensor (DHT22) que mesura tant la temperatura com la humitat de l'aire. Els tres sensors es connecten a un microcontrolador (Arduino Uno), que s'encarrega de processar les mesures per a que siguin transmeses. Les mesures s'envien als nanosatèl·lits mitjançant un terminal utilitzant el protocol KISS (de l'anglès *Keep It Simple, Stupid!*).

S'ha desenvolupat un prototip capaç de recollir les mesures dels sensors y emmagatzemar-les en trames KISS lo més curtes possible. Degut a circumstàncies excepcionals, no s'ha pogut tenir accés als nanosatèl·lits, de manera que la comunicació s'ha comprovat entre dos nodes "terrestres". Queda pendent verificar la comunicació amb els nanosatèl·lits. El prototip és funcional i es deixa preparat per a quan sigui possible tenir accés als nanosatèl·lits. Tot i que falta avaluar la comunicació amb els nanosatèl·lits, que és una part fonamental de l'estudi, el treball de selecció de cultius, construcció del prototip, recollida de dades i transmissió de la informació es va realitzar satisfactòriament.

Abstract

The main objective of this project is to validate the nanosatellites as a technology for crop monitoring. In order to validate the technology, the idea is to design and build a preliminary prototype for a proof of concept, receiving data from a crop and processing it so it can be transmitted to a nanosatellite.

To contextualize the proof of concept, the first step was to do a selection of crops of interest and parameters to measure. After comparing data and considering many high production crops, soy and tomato were chosen. Both are produced in big quantities and in different parts from both northern and southern hemisphere. This is an interesting fact, because the nanosatellites can be used during more time through the year. Regarding the parameters, temperature and humidity were chosen.

The proof of concept consists in building a prototype with three sensors, able to transmit measurements to a nanosatellite. A sensor to measure soil temperature (THERM200), a sensor to measure soil humidity (VH400) and a sensor to measure both air temperature and air humidity (DHT22) were chosen. This three sensors connect to a microcontroller (Arduino Uno), which processes the measurements and prepare them to be transmitted. The measurements are sent to the nanosatellites through a terminal using the KISS (*Keep It Simle, Stupid!*) protocol.

A prototype that receives and stores the measurements from the sensors was successfully built. The measurements are stored in KISS frames with the smallest possible size. Due to exceptional circumstances, it was not possible to have access to nanosattelites. For that reason, the communication was tested between two "terrestrial" nodes". Communication with nanosatellites is pending to be verified. The prototype is functional and ready for when is possible to access to nanosatellites. Although the commnication with nanosatellites was not assessed, which is an important part of the study, the work in selecting crops, building the prototype, gathering data and transmitting the information was done successfully.

Índice

Índice de figuras

Índice de tablas

Agradecimientos

Gracias a Rafael por darme la oportunidad de hacer este proyecto, ayudarme en el uso de Arduino, aclarar dudas en la elaboración de los códigos y prestarme material para hacer el prototipo y calibrado de sensores.

A David, quien puso el proyecto a disposición de la UPC, dedicó su tiempo en reuniones para hablar sobre los nanosatélites y la compañía Aistech Space, y nos acompañó en varias etapas del proyecto.

A Marcos, por guiarme en la etapa final de matizar detalles en los códigos y en el proceso de calibración de sensores.

A Clevis, por prestar sus conocimientos en todo lo relacionado a la selección de cultivos y parámetros de interés.

A Paulina, que nos ayudó a conseguir los sensores analógicos con rapidez.

A Javier, quien dedicó tiempo a comprobar el funcionamiento de dichos sensores.

A Mercé, quien me recibió con brazos abiertos para tomar parte en este máster.

A mi familia, por apoyarme siempre y animarme a seguir adelante en todo momento.

A Rebecca, por su buen corazón y estar siempre a mi lado.

A Dios, por regalarme todas estas bendiciones.

1. Introducción

La tecnificación en el sector agrícola es uno de los aspectos fundamentales a la hora de aplicar nuevas técnicas de cultivo y llevar a cabo su monitorización. Con el paso de estas últimas décadas, el trabajo en la agricultura ha incorporado vehículos especializados para el sector como los tractores agrarios o motocultores. Además, el uso de los robots también está en crecimiento. No sólo podemos encontrar diferentes robots automatizados que se encargan de aplicar pesticidas, recoger frutos en la temporada de cosecha o analizar mediante sensores ópticos el estado de las plantas; también hay robots aéreos que son útiles para la detección de patógenos o para identificar zonas con déficit de irrigación, entre otras aplicaciones (Roldán *et al.*, 2017).

De todos los productos tecnológicos que forman parte de la tecnificación de los cultivos, los dos grupos que nos ocupan en este proyecto son los satélites, concretamente los nanosatélites, y los sensores tanto alámbricos como inalámbricos. Los nanosatélites son un tipo de satélites cuya masa está comprendida entre 1 y 10 kg, y con unas prestaciones y costes de producción menores en comparación a aquellos de mayor tamaño y masa (Buchen, 2014). La utilidad de los satélites en general recae tanto en el intercambio de datos como en la toma de imágenes para analizar diferentes parámetros de interés. Por ejemplo, uno de los usos que un satélite puede tener es el de tomar imágenes de un campo de cultivo para su posterior análisis de salinidad, humedad y pH del suelo (Ghazali *et al*. 2019).

En cuanto a los sensores, su uso en el ámbito agrícola se ha extendido bastante, ya sea en campos de cultivos, invernaderos o laboratorios dedicados al cultivo en condiciones totalmente controladas. Es habitual usar sensores para tomar medidas de temperatura, humedad, salinidad y pH del suelo, aunque algunos poseen otras aplicaciones como detectar el contenido NPK (nitrógeno, fósforo y potasio) de un suelo mediante un sensor de fibra óptica (Ramane *et al.*, 2018).

Aistech Space es una compañía que, con sus satélites y nanosatélites, aporta servicios de trazado y vigilancia de aviones para las aerolíneas, además de análisis de datos, gestión remota de activos y monitorización de cultivos. Frente a tecnologías inalámbricas de intercambio de datos enfocadas al nivel local, como puede ser ZigBee (Kalra *et al.*,

2002; Wang & Gao, 2016), resulta interesante la propuesta de usar nanosatélites para la recepción y envío de datos a una escala global.

Este proyecto se enfoca en hacer un estudio práctico sobre la viabilidad de los nanosatélites con capacidad de comunicación bidireccional, de cara a monitorizar cultivos. Esta monitorización se conseguiría enviando los datos de cultivos a un nanosatélite y de éste a un centro de control, donde serían procesados. A raíz del procesado, se podrían tomar decisiones relativas a, por ejemplo, el riego o fertilización de los cultivos. Estas acciones podrían incluso automatizarse y realizarse a partir de órdenes enviadas a través de los nanosatélites.

Para lograr este cometido, primero se seleccionarán cultivos y parámetros de interés para contextualizar el proyecto. Con esto como punto de partida, el siguiente paso es hacer una prueba de concepto con sensores que tomen medidas de los parámetros escogidos. Estas medidas, después de ser procesadas y almacenadas, se enviarán a un nanosatélite mediante un terminal. Finalmente, se evaluará la capacidad del nanosatélite de recibir y transmitir estos datos, tanto el tamaño de datos que puede manejar como el tiempo de transmisión.

2. Objetivos

Los objetivos de este proyecto son los siguientes:

- Seleccionar dos cultivos de interés para contextualizar el proyecto y tener una idea de las oportunidades a nivel internacional para el uso de los nanosatélites como tecnología de monitorización de cultivos.
- Definir qué parámetros, junto con sus rangos de valores, se van a monitorizar de cara a la elaboración de la prueba de concepto, tomando como referencia los cultivos de interés seleccionados.
- A partir de los parámetros y rangos de valores definidos, buscar sensores adecuados con los cuales implementar la prueba de concepto.
- Elaborar un código que almacene las medidas de cada sensor y las codifique en el menor tamaño posible, para luego ser enviado a un nanosatélite desde un terminal. Otro requisito es que el formato del paquete de datos ha de ser compatible con el utilizado por el terminal.
- Considerar la necesidad de calibración de los sensores y, en su caso, proponer cómo incorporar la calibración al sistema. El cumplimiento de este objetivo quedará condicionada a la disponibilidad de patrones de calibración.

3. Selección de cultivos de interés

3.1. Claves en la búsqueda de cultivos concretos

Para llevar a cabo la selección de cultivos de interés, se han de tener en cuenta las ventajas que aporta la empresa Aistech Space con sus nanosatélites. Por un lado, es una herramienta más para la monitorización de cultivos a distancia y, por el otro, esta tecnología cubre la práctica totalidad de los dos hemisferios de la Tierra. El mero hecho de que los nanosatélites puedan recibir y transmitir datos a lo largo de un recorrido que cubre ambos hemisferios, resulta en una oportunidad para prestar servicios a muchos clientes y empresas con tierras de cultivos repartidas en diferentes países o regiones.

Por ello, conviene estudiar y hallar cultivos producidos a nivel mundial y cuyos mercados estén en crecimiento. Asimismo, es esencial aportar servicios a clientes durante la mayor parte del año. La mayoría de cultivos están sujetos a una estacionalidad fija según la región, clima y hemisferio. Es por ello que, para evitar franjas temporales del año en el que no haya servicios a prestar o haya poca necesidad de monitoreo, se ha de mirar más allá de España e incluso de Europa.

Se optó por hacer un pequeño estudio de algunos de los cultivos más productivos, para tener una idea más clara del terreno que cubren y los beneficios que aportan a nivel económico. Otra pauta para seleccionar cultivos es que se produzcan en diferentes partes del mundo, tanto en el hemisferio norte como en el sur. Con estas claves en mente, se han buscado datos de producción y exportación en FAOSTAT (Food and Agriculture Organization of the United Nations [FAO], 2019) y Tridge (2019), respectivamente. Estos son los cultivos por los que se buscaron datos, que cumplen con al menos dos de las indicaciones mencionadas anteriormente:

- Arroz
- Maíz
- Bananos
- Soja
- Zanahoria y nabos
- Tomate

- Patata
- Trigo

3.2. Selección final de cultivos

Teniendo en cuenta las claves descritas en el anterior apartado, se procedió a hacer una selección final de los cultivos de interés para su potencial monitorización mediante nanosatélites. De entre todos los de la lista presentada anteriormente, se optó por la soja y el tomate.

Hay diversas razonas que han motivado esta elección. En primer lugar, tanto el tomate como la soja tienen una demanda alta y están en crecimiento (FAO, 2019), además de tener un valor de exportación que supera el billón de dólares (Tridge, 2019). En segundo lugar, los dos productos se cultivan en grandes cantidades y en diferentes partes del mundo de ambos hemisferios. Por último, se escogieron estos dos productos por cuestión de intereses personales.

3.2.1. Datos de producción, exportación y área cosechada para el cultivo de soja

La **Figura 3-1** muestra un mapa mundial con la producción media anual de soja por países entre 2013 y 2017.

La mayor parte de la producción recae en América y Asia, especialmente en países como Estados Unidos (108 millones de toneladas de media), Brasil (95 millones de toneladas), Argentina (55 millones de toneladas), China (12 millones de toneladas) e India (11 millones de toneladas). Esto resulta interesante porque hay referentes en producción de soja tanto en América del Norte como América del Sur, además de Asia.

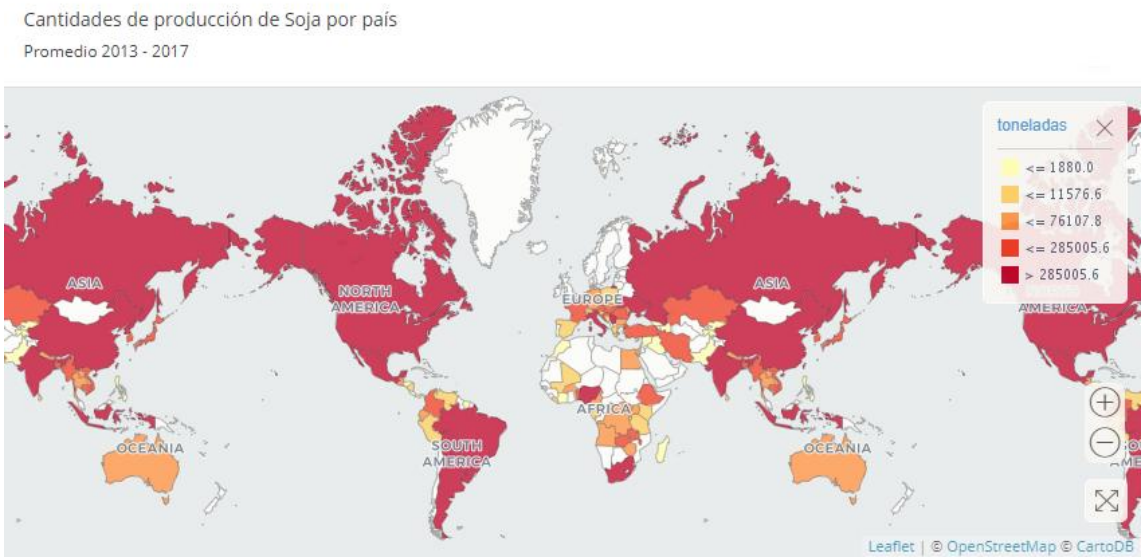

Figura 3-1. Producción media de soja en toneladas por país, desde el año 2013 hasta el 2017 (FAO, 2019).

Si nos fijamos en los datos de exportación según Tridge (2019), una vez más Brasil y Estados Unidos son los dos países con los números más altos, con 68 y 55 millones de toneladas exportadas en el año 2017, respectivamente. No sólo eso, sino que el valor total de exportación supera los 20 billones de dólares en ambos casos. Por lo tanto, se puede concluir que ambos países no son solamente grandes productores, sino también proveedores para otros países del mundo.

La **Figura 3-2** muestra la evolución de la producción total y área cosechada del cultivo de soja en todo el mundo entre 2013 y 2017.

Se puede observar de manera clara que hay una tendencia a crecer en los próximos años, tanto en producción como en área cosechada. Esto indica posibles oportunidades de cara al futuro. Es de esperar que nuevos productores se adentren en el cultivo de este producto para sacar provecho de la demanda creciente. Esta circunstancia presenta una buena oportunidad para la oferta de servicios para tecnificar los cultivos.

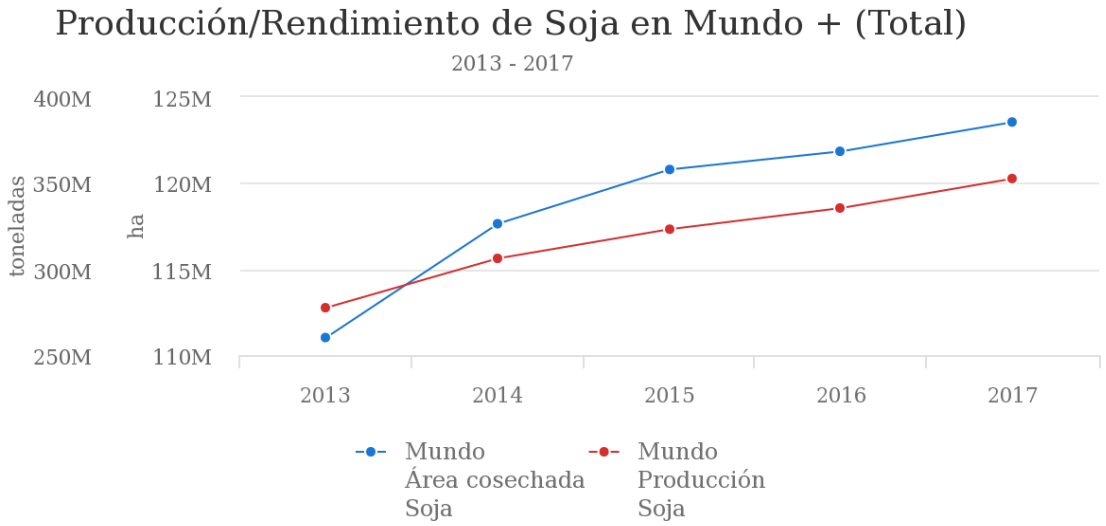

Figura 3-2. Evolución de la producción total y área cosechada de la soja en todo el mundo entre 2013 y 2017 (FAO, 2019).

3.2.2. Datos de producción, exportación y área cosechada para el cultivo de tomate

La **Figura 3-3** muestra un mapa con la producción media anual de tomate por país entre 2013 y 2017.

En el caso del tomate, la mayor productora es China con unos 55 millones de toneladas de producción media. Detrás de ella le siguen India (18,5 millones de toneladas de media) y Estados Unidos (13,6 millones de toneladas), pero resulta interesante la inclusión de Egipto (7,79 millones de toneladas), Italia (6,96 millones de toneladas), España (4,78 millones de toneladas) y México (3,78 millones de toneladas) como parte de las 10 grandes productoras de tomate a nivel mundial.

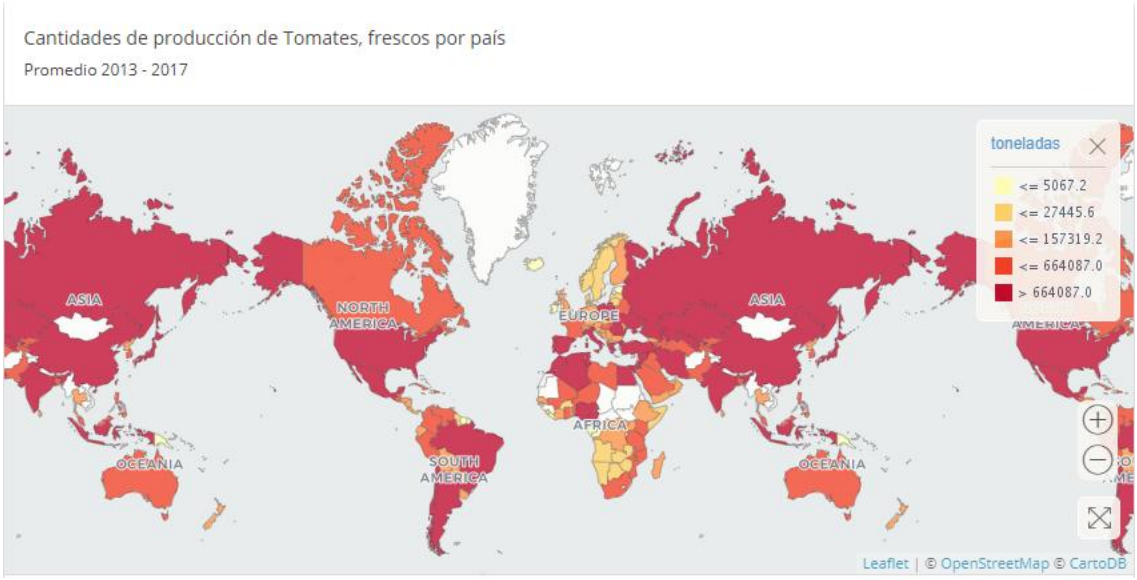

Figura 3-3. Producción media anual en toneladas de tomate por país entre 2013 y 2017 (FAO, 2019).

Los datos de Tridge (2019) colocan a México y España como dos de los tres países más importantes en la exportación del tomate, junto con Holanda. Por un lado, México exportó en el año 2017 1,74 millones de tonelada, dos quintas partes de la producción de ese año, con un valor total de 1,94 billones de dólares. Por otro lado, España exportó 810 miles de toneladas por un valor de 1,13 billones de dólares.

Finalmente, la **Figura 3-4** muestra la evolución de la producción total y la del área cosechada del cultivo del tomate en todo el mundo entre 2013 y 2017.

El primer detalle que salta a la vista es la caída pronunciada en área cosechada que se produce entre el 2014 y 2015. Después vuelve a crecer otra vez, para retomar prácticamente el mismo valor que tenía en 2013. En cuando a la producción total, se aprecia una subida constante con el paso de los años, con vista a seguir creciendo de cara al futuro.

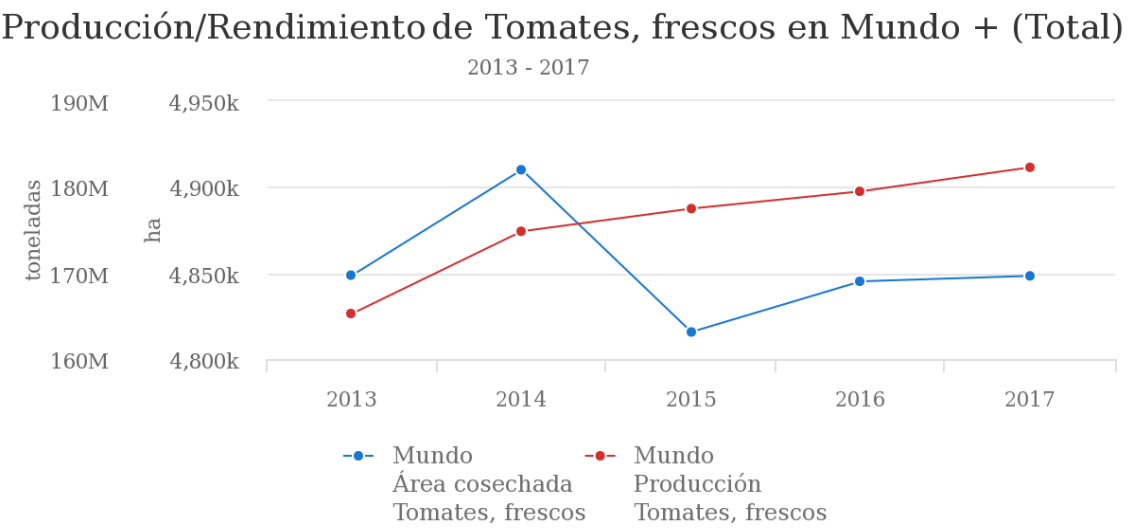

Figura 3-4. Evolución de la producción total y área cosechada del tomate en todo el mundo entre 2013 y 2017 (FAO, 2019).

4. Parámetros de interés y búsqueda de los sensores para la prueba de concepto

4.1. Parámetros de interés y rango de valores

Elegidos los dos cultivos de interés, el tomate y la soja, el siguiente paso a dar es establecer qué parámetros se van a monitorizar y, sobre todo, el rango de valores que conviene manejar. Todo ello permitirá acotar la búsqueda de los sensores a utilizar en la prueba de concepto.

Uno de los parámetros más importantes que definen el crecimiento de un cultivo determinado es la temperatura. Para el caso del tomate, un rango de temperaturas de entre 16 ºC y 32 ºC es adecuado. Para el proceso de germinación, lo óptimo es manejar temperaturas frescas y que, conforme el cultivo crezca y dé frutos, la temperatura vaya ligeramente en aumento (Adams *et al.*, 2001; Andrades & Martínez, 2014). Por esa razón, si hablamos del hemisferio norte , es habitual que la temporada de plantar tomates comience en marzo y abril. Algunos estudios, como el de Piper & Boote (1999), determinan un rango de temperaturas óptimo similar para el caso de la soja, con preferencia a mantenerse entre 22 ºC y 28 ºC.

El rango de valores debe ir más allá de de las temperaturas óptimas. Como la prueba de concepto se realizará en España, conviene que los sensores de temperatura puedan medir desde -10 ºC hasta 50 ºC para no estar limitados por la estacionalidad. Además, este rango de valores es más que suficiente tanto para la monitorización del tomate como de la soja.

Otro parámetro que es importante es la humedad del aire o cantidad de agua en el suelo. Los cultivos necesitan del agua no sólo por ser una de las fuentes principales para su adecuado crecimiento, sino porque su presencia facilita la captación de nutrientes. En este caso, es preferible que un sensor de humedad maneje un rango entre 0 % y 100 % de humedad relativa.

Aparte de la temperatura y la humedad, también hay otros parámetros como el pH, la salinidad o conductividad eléctrica, el contenido NPK y la capacidad de intercambio

catiónico. En particular, el pH resulta útil monitorizarlo puesto que nos da una idea de la disponibilidad de los nutrientes en el suelo. El rango más adecuado para la soja gira en torno a un pH entre 6 y 7 (McGrath *et al.*, 2013), mientras que para el tomate es entre 6,5 y 7 (Andrades & Martínez, 2014). Ambos cultivos son tolerantes a suelos ligeramente más ácidos o alcalinos, pero se pueden apreciar pérdidas de rendimiento a causa de la menor disponibilidad de nutrientes. Por esa razón, no es habitual que un campo de cultivo se aleje más allá de un rango de pH entre 5 y 9, por lo que un sensor que pueda medir este rango de pH es más que suficiente.

Analizar el contenido de materia orgánica o NPK (nitrógeno, fósforo y potasio) del suelo, así como saber la concentración de otros elementos como el calcio, magnesio o sodio, es también otro factor crucial para conocer la fertilidad de un suelo. Por ejemplo, la compañía Starke Ayres (2014) presenta algunos datos sobre el rango de concentraciones ideales de algunos elementos (**Tabla 4-1**) para que el tomate tenga suficientes nutrientes a lo largo de las etapas de su crecimiento:

Tabla 4-1. Concentración ideal de algunos elementos en el suelo para asegurar fertilidad en la producción de tomate (Starke Ayres, 2014).

Elemento	Concentración (mg/Kg)
P	30 - 60
K	100 - 250
Ca	300 - 2000
Mg	120 - 300
Na	10 - 50

Por último, otro parámetro que resulta interesante es el GDD (*growing degree-days*). Se trata de un dato acumulativo que tiene en cuenta la temperatura máxima y mínima de cada día, para luego hacer un sumatorio de estos resultados a lo largo de las etapas de crecimiento. A continuación, se puede ver una de las ecuaciones que sirven para calcular el GDD de un determinado día (Gordon & Bootsma, 1993):

$$GDD = \frac{Tmax + Tmin}{2} - Tbase$$

El *Tbase* o temperatura base se refiere a la temperatura mínima en la cual se considera que un cultivo puede comenzar a crecer. Cada uno tiene su propia temperatura base. En el caso del tomate y la soja, se ha establecido que la temperatura base es de 10 ºC (Gordon & Bootsma, 1993). Conforme pasan los días, se acumulan los resultados del cálculo de GDD y se toma como referencia para predecir a grandes rasgos las etapas de crecimiento, la necesidad de usar pesticidas, prever una maduración temprana o tardía, etc. Si bien es un dato útil, conviene usar este parámetro en cooperación con otros, ya que el rendimiento de un cultivo está afectado por muchos factores más que la temperatura.

Tras una serie de reuniones con Aistech Space, decidimos buscar sensores que midan temperatura o humedad, tanto del aire como del suelo. Los sensores deberán ser económicos, fáciles de usar y con prestaciones acorde a los parámetros escogidos. En el caso de sensores de temperatura, con poder medir un rango entre -10 ºC y 50 ºC y tener una resolución de 0,1 ºC es más que suficiente. En cuanto a los sensores de humedad, es importante que el rango sea entre 0 % y 100 % en humedad relativa, pero no es necesario que la resolución sea mayor de 1 %.

4.2. Búsqueda de sensores

Hay distintos tipos de sensores que se utilizan para monitorizar los cultivos. Las prestaciones de estos sensores pueden variar según las necesidades de la persona o empresa, siendo uno de los factores más importantes el nivel de control que se quiera tener sobre los parámetros que afectan al crecimiento de un cultivo.

En el mercado hay una serie de sensores disponibles que bien pueden medir un solo parámetro, como puede ser la temperatura, o medir un conjunto de parámetros que pueden estar interrelacionados entre sí o ser independientes.

Como se indicó en el apartado anterior, se buscan sensores económicos y fáciles de usar, pues sólo se pretende hacer una prueba de concepto y no crear un producto comercial. Con esto en mente, tras hacer una búsqueda y encontrar productos con

prestaciones variadas, se escogieron los sensores DHT22, THERM200 y VH400 para la prueba de concepto, y el SHT10 para la calibración de los sensores analógicos. Estos cuatro sensores se presentan en la siguiente sección de la memoria.

5. Materiales y métodos

5.1. Sensores utilizados

5.1.1. DHT22

DHT22 (**Figura 5-1**), también conocido como AM2302, es un sensor digital que permite medir la temperatura y la humedad en el aire de manera simultánea (Adafruit Industries, 2019a). Para la medición de temperatura hace uso de un termistor, mientras que para la humedad utiliza un sensor capacitivo. Tiene dimensiones pequeñas y, en la versión que se ha usado para este proyecto, posee tres pines: Vcc, Output y Gnd, que son la alimentación de sensor, envío de datos y toma de tierra, respectivamente.

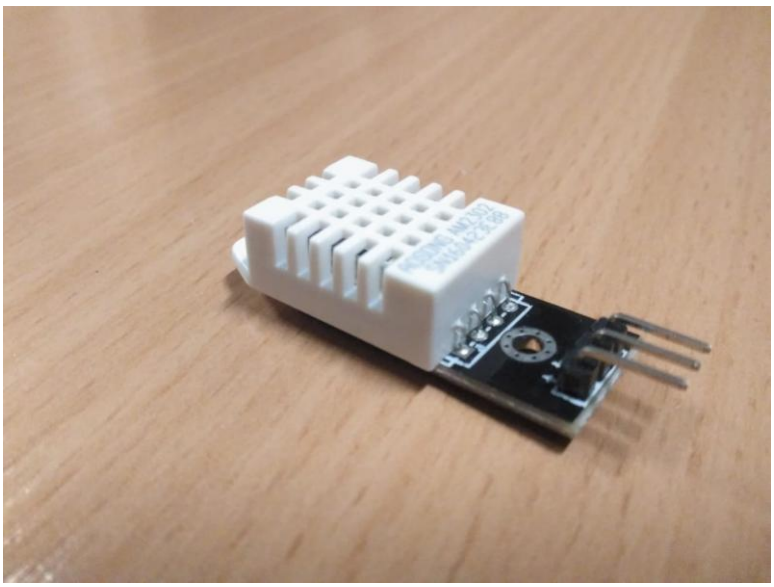

Figura 5-1. Foto del sensor de temperatura y humedad DHT22.

La **Tabla 5-1** muestra algunas especificaciones del sensor DHT22, con datos sobre su rango de medida, precisión, resolución e histéresis:

Tabla 5-1. Algunas de las especificaciones del sensor DHT22 (Adafruit Industries, 2019a).

Parámetro	Humedad (%HR)	Temperatura (ºC)
Rango de medida	0 a 100	-40 a 80
Precisión	De ±2,0 a ±5,0	±0,5
Resolución	0,1	0,1
Histéresis	±0,3	-

El sensor puede medir temperaturas entre -40 ºC y 80 ºC, un rango más amplio que el comentado anteriormente, y humedad entre 0 % y 100 %, tal y como se estableció en la sección previa. La resolución para ambos parámetros cumple con lo que se necesita de cara a la prueba de concepto. Incluso teniendo en cuenta la precisión e histéresis que presentan las lecturas de humedad, el DHT22 es un sensor adecuado para captar medidas en el aire en el prototipo del proyecto.

Otro detalle a tener en cuenta es que la frecuencia de muestreo de DHT22 es de 0,5 Hz como máximo. Eso implica que los datos tendrán 2 segundos de antigüedad. Eso no es un inconveniente, ya que la temperatura y la humedad en los campos de cultivo no cambian bruscamente de un segundo a otro, sino que se tardan minutos e incluso horas hasta tener una diferencia perceptible para el cultivo.

5.1.2. THERM200

El sensor THERM200 (**Figura 5-2**), de la compañía Vegetronix, es uno de los dos sensores analógicos que se han utilizado en este proyecto. La finalidad principal de éste es la de medir la temperatura del suelo para un momento determinado, aunque también puede medir la temperatura del aire. La manera de calcular la temperatura es mediante una relación lineal con los cambios de voltaje del sensor, de 0 a 3 V (Vegetronix, 2019a).

Figura 5-2. Foto del sensor de temperatura en suelo THERM200, de la compañía Vegetronix.

La **Tabla 5-2** muestra algunas de las especificaciones de THERM200 en relación al rango de medida, la exactitud y la resolución:

Tabla 5-2. Algunas especificaciones del sensor THERM200 (Vegetronix, 2019a).

Parámetro	**Temperatura (ºC)**
Rango de medida	De -40 a 85
Exactitud	±0,5
Resolución	0,125

Una gran ventaja que aporta este sensor es que, al ser totalmente impermeable, no se daña en caso de entrar en contacto con agua. Comparado con el sensor DHT22, tiene un rango de medida, precisión y resolución bastante similares, por lo que también cumple con las indicaciones para la prueba de concepto. Sin embargo, tiene una respuesta lenta y no puede enviar lecturas estables cuando la temperatura cambia muy rápido. Tarda unos 15 minutos para que el centro del sensor esté a la misma temperatura que la del

exterior. Eso no es un punto negativo para este proyecto, debido a que la temperatura del suelo en los cultivos siempre cambia con lentitud. Por último, para leer correctamente la temperatura, el sensor debe estar totalmente enterrado bajo suelo.

Aunque la compañía Vegetronix aporta sus propias curvas de voltaje y temperatura para este sensor, es preciso hacer un calibrado para asegurarnos de que las medidas de THERM200 se correspondan con la temperatura real.

5.1.3. VH400

El sensor analógico VH400 (**Figura 5-3**), también de la compañía Vegetronix, mide la humedad en el suelo. El sensor debe estar en contacto con un líquido o sólido; en caso contrario, siempre tendrá una lectura de 0 V, lo que implica un 0 % de humedad. Según la compañía, este sensor no calcula la humedad mediante una relación lineal, sino que existen cuatro curvas de relación voltio-humedad hasta alcanzar el 50 % de humedad. Si en la lectura se supera ese porcentaje, existe una quinta curva que sirve como aproximación hasta el 100 % (Vegetronix, 2019b). Es preciso hacer un calibrado para saber si esta relación voltaje-humedad se corresponde con la realidad.

Figura 5-3. Foto del sensor de humedad en suelo VH400, de la compañía Vegetronix.

La **Tabla 5-3** muestra algunas de las características del sensor y los rangos de medidas para VH400:

Tabla 5-3. Algunas especificaciones del sensor VH400 (Vegetronix, 2019a).

Parámetro	Humedad (%HR)
Rango de medida	De 0 a 100
Exactitud (a 25 ºC)	±2,0
Resolución	0,1

Tanto el rango de medida como la resolución cumplen con lo que se necesita para la prueba de concepto y para monitorizar los cultivos seleccionados. La exactitud es algo mejor que la del DHT22, pero está indicado que es sólo a 25 ºC, sin especificar cómo varía la exactitud a diferentes temperaturas. Este es un motivo más por el que convendría hacer un calibrado del sensor.

Igual que con THERM200, este sensor es totalmente impermeable pero también muy resistente a la corrosión. Contrario al anterior sensor analógico, las lecturas no se desestabilizan tanto con los cambios bruscos de humedad, ya que tiene un tiempo de respuesta de 400 milisegundos. Las variaciones de humedad nunca son tan rápidas en los cultivos de soja y tomate, por lo que nos sirve. Asimismo, es recomendado enterrar el sensor entero, aunque sólo es necesario que toda la hoja verde esté en contacto con el suelo que se quiere medir.

5.1.4. SHT10

El sensor digital SHT10 (**Figura 5-4**) es el último que forma parte del proyecto y mide la temperatura y humedad tanto la del aire como la del suelo. Se usará como referencia en el calibrado de los sensores analógicos THERM200 y VH400. Gracias al encapsulado impermeable con revestimiento metálico, no deja pasar nada más que el aire. Aun así, no se recomienda dejarlo en agua por más de una hora seguida (Sensirion, 2020).

Figura 5-4. Foto del sensor de temperatura y humedad SHT10 de la compañía Sensirion, utilizado para un proceso de calibrado en los sensores analógicos.

La **Tabla 5-4** da a conocer las especificaciones respecto al rango de medida, exactitud y resolución del sensor SHT10 de Sensirion:

Tabla 5-4. Algunas especificaciones del sensor SHT10 (DFRobot, 2020).

Parámetro	Humedad (%HR)	Temperatura (ºC)
Rango de medida	0 a 100	-10 a 80
Exactitud	±5,0	±0,5
Resolución	0,1	0,1

Este sensor posee un rango de medida de temperatura menor que los del DHT22 y del THERM200, y la exactitud para la medición de humedad es más baja respecto al VH400. El SHT10 tiene un tiempo de respuesta menor de 8 segundos, lo cual es una ventaja de cara al sensor analógico THERM200, pero no de cara a VH400. De todos modos, este

tiempo de respuesta no afectará negativamente al calibrado, pues se harán medidas a lo largo de una hora a temperatura estable.

Para un buen calibrado de los sensores analógicos, es necesario que el sensor de referencia tenga mejor precisión y resolución. Debido a diferentes circunstancias, no fue posible tener un sensor con mejores prestaciones.

5.2. Arduino Uno y Arduino IDE

Seleccionados los sensores para la prueba de concepto, utilizaremos un Arduino Uno para controlar los sensores, recoger sus medidas y darles el formato adecuado para enviarlas a un nanosatélite.

Arduino Uno (**Figura 5-5**) es uno de los microcontroladores de la familia Arduino que existen en el mercado. Es considerado el más sencillo para un primer contacto con este tipo de microcontroladores. Este dispositivo dispone de 14 salidas y entradas digitales, 6 salidas PWM, conexión USB y 6 entradas analógicas con conversor AD.

Figura 5-5. Imagen del microcontrolador Arduino Uno, en su versión estándar.

Asimismo, para conectar Arduino Uno con los sensores en este proyecto, ha sido necesario usar una serie de cables con pines macho y hembra para conectarlos entre ellos, resistencias y *breadboards* o placa de prototipos.

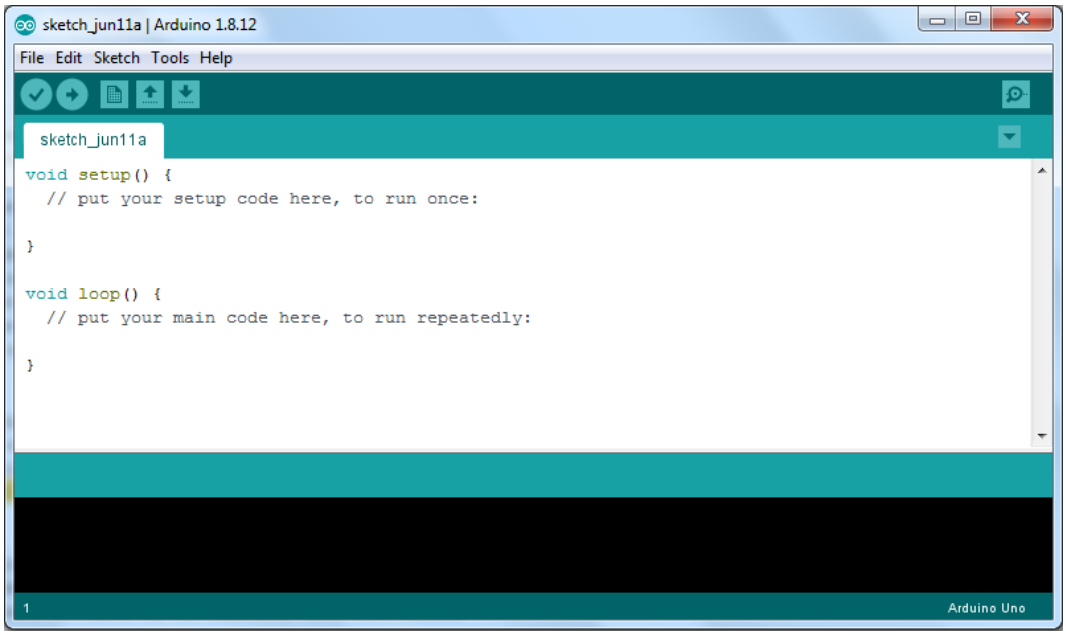

Figura 5-6. Captura de pantalla del programa Arduino IDE.

La **Figura 5-6** muestra una captura de pantalla de Arduino IDE (*Integrated Development Environment*), un programa de código abierto que permite la programación de Arduino Uno. Para ello, los códigos se escriben en hojas denominadas *sketches*, siempre en lenguaje C y C++ (Bayle, 2013).

La aplicación es bastante versátil, poco exigente en cuanto a requisitos de ordenador y soporta el uso de librerías, las cuales se explican en el apartado siguiente.

5.3. Librerías de Arduino utilizadas

Las librerías de Arduino son archivos que o bien añaden funcionalidades extras a la hoja de Arduino IDE en el que se trabaja o permiten simplificar algunos procedimientos y escrituras de código (Bayle, 2013). Por ello, son una herramienta muy útil en la elaboración de los programas. Gracias a estas librerías, funcionalidades complicadas de programar desde cero son accesibles incluso para un programador con conocimientos básicos.

En este proyecto se han utilizado tres librerías distintas: DHT, CRC32 y SHT1x.

5.3.1. Librería DHT

Esta librería sirve de puente comunicar los sensores del tipo DHTxx y el Arduino Uno. Para el caso de este proyecto, la finalidad de la librería DHT es la de leer los datos que envía el DHT22 al monitorizar la temperatura y humedad en tiempo real, para luego transformarlo en valores cuantificables y convertirlos en las unidades correspondientes (Adafruit Industries, 2019b).

Implementa asimismo una función que calcula el índice de calor, a partir de los datos tomados de la temperatura y humedad. Por ello, se incluyó también este valor como parte de los datos a enviar.

5.3.2. Librería CRC32

Comparado con el anterior, el caso de la librería CRC32 (Baker, 2019) es un tanto distinto. En vez de interactuar con lecturas de un sensor, se encarga de recoger un valor o grupo de valores (usualmente una matriz de datos) para luego calcular el checksum.

La finalidad del checksum es la de asegurar la integridad de los datos que se envían de un terminal a otro para detectar así cambios accidentales, debido a errores en la comunicación, y evitar que se procesen valores incorrectos.

5.3.3. Librería SHT1x

La librería SHT1x (Oxer & Ribble, 2011) juega el mismo papel que la de DHT, pero en este caso para los sensores SHT1x. Por ello, la función de esta librería es recibir las lecturas enviadas por el sensor SHT10 y convertirlas en los valores y unidades de temperatura y humedad correspondientes.

5.4. Protocolo KISS

Uno de los objetivos de este proyecto es la de hacer una prueba de concepto en el que se envíen datos desde una terminal a un nanosatélite. En nuestro caso particular, el tamaño máximo de datos que el nanosatélite puede transmitir en un instante es de 192 bytes. La elaboración del código se ha de llevar a cabo teniendo en mente este límite,

para poder enviar la mayor cantidad de datos posibles, sin sobrepasar este número de bytes.

Además, es necesario utilizar un protocolo que permita verificar los datos enviados, así como detectar su inicio y final. El protocolo elegido fue KISS, del inglés *Keep It Simple, Stupid* (Chepponis & Karn, 1987).

Este protocolo utiliza una codificación hexadecimal de la información. La codificación hexadecimal usa números del 0 al 9 y letras de la A hasta la F. Los caracteres del 0 al 9 corresponden a los mismos números en valor decimal, mientras que de la A a la F se abarca un rango de valores decimales entre 10 y 15. Estos caracteres también se pueden combinar, de manera que podemos obtener un valor mayor de 15. Cada carácter hexadecimal tiene 4 bits de tamaño, por lo que dos caracteres son 1 byte.

KISS utiliza caracteres especiales que marcan el inicio y final de un envío de información, dando lugar a lo que se conoce como trama. Además de estos caracteres, hay otros destinados a evitar la confusión de estos caracteres con otros de información que contengan el mismo valor. La **Tabla 5-5** muestra cuál es el significado de cada uno de los valores hexadecimales usados para el protocolo KISS.

Tabla 5-5. Significado de cada valor hexadecimal en el protocolo KISS (Chepponis & Karn, 1987).

Valor hexadecimal	Abreviación	Descripción
C0	FEND	Frame End
DB	FESC	Frame Escape
DC	TFEND	Transposed Frame End
DD	TFESC	Transposed Frame Escape

El valor C0, referido a FEND o *Frame End*, es el encargado de marcar el inicio y final de una trama. Los demás valores sirven para sustituir valores de los datos que coincidan tanto con el valor C0, para sustituirlo por DB+DC, como con DB, para sustituirlo por DB+DD. De este modo, el receptor sabe con seguridad dónde empieza y acaba una trama.

En resumen, lo que se pretende con el protocolo KISS es enviar una trama con la siguiente estructura:

- FEND como inicio de trama.
- Datos.
- Checksum.
- FEND como final de trama.

5.5. Fritzing

Fritzing es un programa de código abierto que sirve para realizar esquemas de circuitos electrónicos, así como representaciones de su montaje (Bayle, 2013). Su enfoque es el de ayudarnos a entender cómo podemos montar un circuito determinado y entender mejor qué conexiones, cables, placa de prototipos, resistencias, etc., son necesarios.

En la **Figura 5-7** se puede ver una imagen con la interfaz de Fritzing. En este proyecto se ha utilizado este programa para ilustrar el montaje de la prueba de concepto del proyecto, así como uno de los montajes para el calibrado de los sensores analógicos.

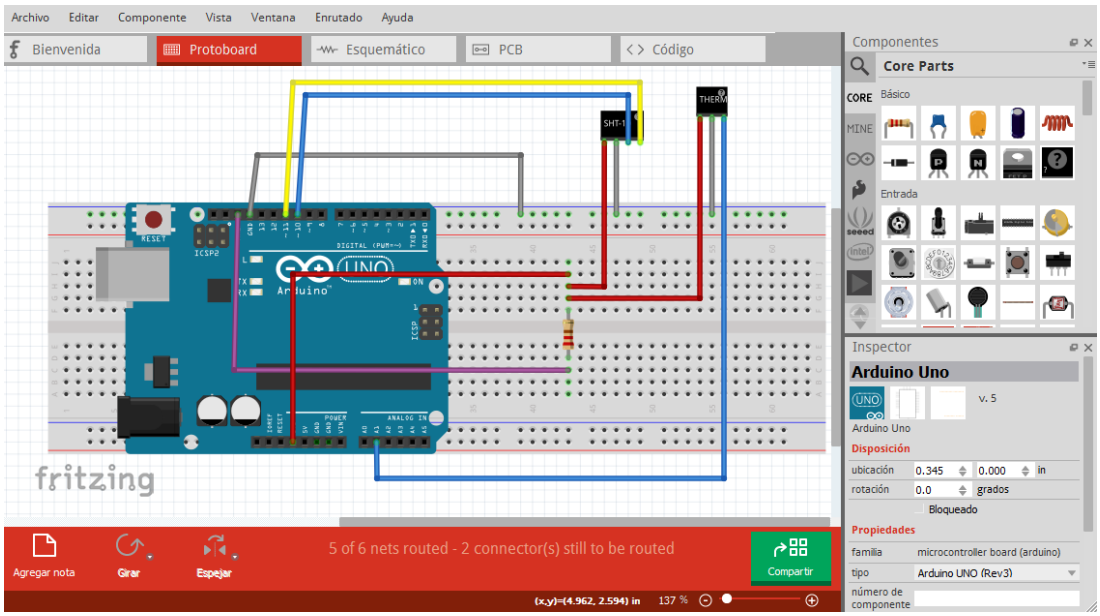

Figura 5-7. Captura de pantalla de la interfaz de Fritzing, con la vista Protoboard (placa de prototipos) del programa.

6. Resultados

6.1. Prototipo de los sensores y Arduino Uno

En primer lugar, para llevar a cabo la lectura de las medidas es necesario hacer un montaje en el que se conecten los sensores analógicos THERM200 y VH400, y el sensor digital DHT22 en un mismo Arduino Uno. Para cumplir el objetivo de este proyecto, se deben recoger lecturas de los tres sensores al mismo tiempo, y no por separado, para así enviar una sola trama KISS que recoja los datos de todos los sensores para cada intervalo de tiempo.

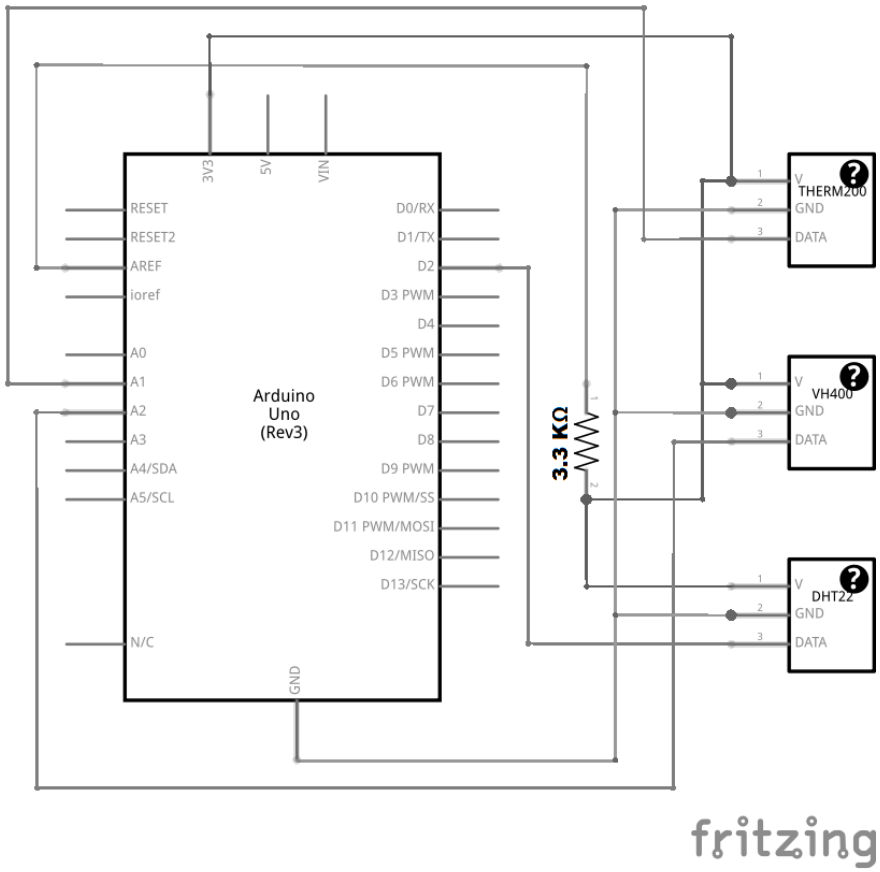

Figura 6-1. Esquema del prototipo con los tres sensores, hecho mediante el programa Fritzing .

El montaje del prototipo se hizo con la ayuda de una *breadboard* (placa de prototipos). En la **Figura 6-1** se puede apreciar que se ha colocado una resistencia de 3,3 kΩ como parte del circuito, en la misma línea en la que se conectan los cables de alimentación (V)

de los tres sensores. Estos sensores hacen una lectura en base a una variación de tensión entre 0 y 3 V. La resistencia de 3,3 kΩ, junto con la resistencia de entrada de AREF, sirve para obtener una tensión de referencia (VREF) de 3 V a partir de los 3,3 V de alimentación del Arduino Uno.

Las lecturas de los sensores analógicos presentan fluctuaciones debido al ruido de la tensión de alimentación. Para reducir el ruido se suele utilizar un condensador. Las circunstancias excepcionales de confinamiento durante la realización de este trabajo, impidieron acceder a este material y, como alternativa, se promediaron un elevado número de lecturas para eliminar la fluctuación de la medida.

Otra razón de usar la placa de prototipos es la de poder unificar todas las conexiones a tierra (GND) a una sola, en oposición a conectarlo a tres pines diferentes. De esta manera, los tres sensores comparten la misma conexión, consiguiendo que todos los sensores tengan una misma referencia de voltaje. Aunque la placa de prototipos permite conectar incluso cuatro o más sensores a un mismo Arduino Uno, en caso de que se decidiera ampliar convendría construir un prototipo en una placa PCB.

Por último, las salidas DATA, encargadas en trasmitir a Arduino Uno los datos de lectura de cada sensor, están conectadas a pines diferentes. Estos pines serán referenciados en el código para que el microcontrolador trabaje con las lecturas recibidas. Los pines que usa cada sensor para la transmisión de datos son las siguientes:

- **Salida DATA de THERM200**: pin A1.
- **Salida DATA de VH400**: pin A2.
- **Salida DATA de DHT22**: pin D2.

6.2. Prototipo del proceso de calibrado

El montaje del prototipo para la calibración de los sensores analógicos sigue la misma base que la del prototipo presentado en el apartado anterior. Es más sencillo y necesita de menos conexiones, pues se conectan dos sensores a Arduino Uno en vez de tres. En la **Figura 6-2** se puede ver el montaje que se hizo para calibrar el sensor analógico de temperatura THERM200 con SHT10 como sensor de referencia.

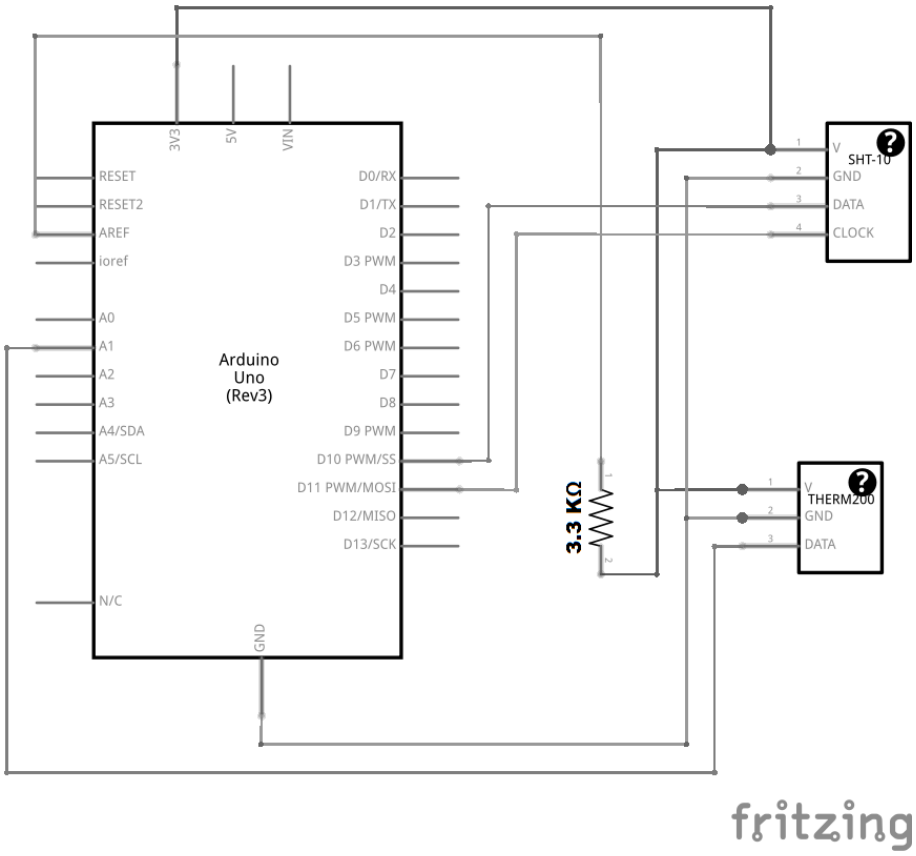

Figura 6-2. Esquema del prototipo para proceso de calibrado, hecho mediante el programa Fritzing. Esta imagen sólo muestra el montaje con SHT10 y THERM200.

En el esquema, además de las salidas V, GND y DATA, se observa que el sensor SHT10 presenta otra llamada CLOCK. Esta salida se usa para sincronizar la comunicación entre el microcontrolador y el sensor SHT10. El resto de salidas que posee dicho sensor son iguales a los de otros sensores de este proyecto.

6.3. Código de lectura y envío de datos

La versión final del código de lectura que se creó para la prueba de concepto se organiza en diferentes bloques, cada uno encargado de cumplir con una función determinada. El código completo se encuentra al final de la memoria, en el **Anexo 1**. En esta sección se describe a grandes rasgos la finalidad de cada bloque, sin entrar en detalle en todas las líneas de código.

```
// Primero incluimos librerías
#include "DHT.h"
#include <CRC32.h>

// Luego definimos algunas variables, empezando por DHT22
#define DHTPIN 2 // Pin digital al que se conecta el sensor DHT22
#define DHTTYPE DHT22 // Determina qué tipo de sensor DHT está conectado
DHT dht(DHTPIN, DHTTYPE); // Carga un algoritmo de la librería DHT según los parámetros definidos
float rawValueDHTTemp; // Variable que almacena la lectura de temperatura del DHT22, antes de conversiones
float rawValueDHTHum; // Variable que almacena la lectura de humedad del DHT22, antes de conversiones
int airTempEntero; // Variable que almacenará la parte entera de la temperatura, tras conversión
byte airTempDecimal; // Variable que almacenará la parte decimal de la temperatura, tras conversión
byte airHum; // Variable que almacenará el valor de la humedad, tras conversión
int indiceCalorEntero; // Variable que almacenará la parte entera del Índice de Calor, tras conversión
byte indiceCalorDecimal; // Variable que almacenará la parte decimal del Índice de Calor, tras conversión
```

Figura 6-3. Primeras líneas de códigos, encargadas de incluir librerías y definir variables de sensores.

El propósito del primer bloque, mostrado en la **Figura 6-3**, es el de permitir que Arduino Uno utilice las librerías DHT y CRC32, así como definir variables que se encargarán de guardar los valores de las lecturas de temperatura y humedad. También se indica en qué pines están conectadas las salidas DATA de cada uno de los sensores.

```
//Para pasar variables float e int a bytes
union TempThermEnteroPasoABytes{
  int TThermByte;
  byte ttb[2];
} ttbin, ttbout;
```

Figura 6-4. Ejemplo de la creación de la variable “union".

El segundo bloque es una estructura para guardar los datos de algunas medidas en un formato compatible con KISS. En la **Figura 6-4** se pueden ver dos de las ocho variables que se crearán en esta estructura. Tras buscar e intentar distintas maneras de separar las variables de 2 o más bytes en bytes individuales, para que los datos ocupen el menor espacio posible, se optó por usar la estructura "union" para lograr este cometido.

Esta estructura toma una variable determinada y la guarda en una matriz de variables nuevas, todas con el tamaño de un byte. Se ha usado la estructura "union" para las partes enteras de las lecturas de temperatura (sensores THERM200 y DHT22) y el índice de calor (sensor DHT22), y se aprovechó para dividir asimismo la variable checksum, de la cual hablamos más adelante, en 4 bytes separados.

```
void setup() {
  Serial.begin(9600);
  analogReference(EXTERNAL); // Necesario para la lectura de los sensores analógicos
  dht.begin(); // Da una orden para activar la lectura de DHT22
}
```

Figura 6-5. Función de “void setup”, encargada de inicializar.

El tercer bloque (**Figura 6-5**) es la función "void setup". Antes de hacer nada más, el microcontrolador primero ejecuta todas las órdenes que vienen incluidas dentro de esta función. A continuación, se explica la función de cada línea:

- **Serial.begin(9600)**: marca la velocidad de transmisión de datos con el puerto serie del Arduino IDE.
- **analogReference(EXTERNAL)**: se informa a Arduino Uno que tome como referencia el voltaje aplicado al pin AREF.
- **dht.begin**: se da una orden al sensor DHT22 de inicializar las medidas.

```
// Determinamos lo que sucederá en cada rotación del bucle
void loop() {
  // BLOQUE DE LECTURAS PARA THERM200, VH400 y DHT22
  rawValueTherm = 0; // Se reinician las cuatro variables
  rawValueHum = 0;
  rawValueDHTTemp = 0;
  rawValueDHTHum = 0;
  for (int i = 0; i < 150; i++) { // Lecturas en cada bucle de todos los sensores
    rawValueTherm += analogRead(thermPin); // Hace una sumatoria de las lecturas del sensor THERM200
    rawValueHum += analogRead(humPin); // Hace una sumatoria de las lecturas del sensor VH400
    rawValueDHTTemp += dht.readTemperature(); // Hace una sumatoria de las lecturas del sensor DHT22
    rawValueDHTHum += dht.readHumidity();
    delay(43); // Se espera X tiempo para almacenar una lectura
  }
```

Figura 6-6. Imagen del primer procedimiento de la función “void loop”.

Tras la función "void setup", sigue la función "void loop" (**Figura 6-6**). El microcontrolador, después de ejecutar las órdenes de la función anterior, ejecuta una y otra vez las instrucciones incluidas en "void loop" mientras el nodo tenga alimentación, no se reinicie o no se cargue un código nuevo.

El primer paso de esta función es la de recibir las lecturas de los sensores y almacenarlas en distintas variables creadas dentro del mismo bucle. En este caso, se recoge una lectura cada 43 milisegundos hasta tener un total de 150 lecturas. El objetivo

de recoger 150 lecturas en total es la de luego promediar las medidas para eliminar las fluctuaciones, tal y como que se comentó en el apartado 6.1.

```
  // Conversión de los valores de las lecturas de THERM200
  float voltajeTherm = (rawValueTherm / 150.0)*(3.0 / 1023.0);
  float valorTherm = 41.67*voltajeTherm-40.0;
  soilTempEntero = int(valorTherm);
  soilTempDecimal = byte((valorTherm-soilTempEntero)*10);
// Conversión de la variable entera de "int" a "byte"
  ttbin.TThermByte = soilTempEntero;
  for(int conv1=0 ; conv1 < sizeof(ttbin.TThermByte) ; conv1++)
      ttbout.ttb[conv1] = ttbin.ttb[conv1];
```

Figura 6-7. Primer bloque de conversión de las lecturas obtenidas de los sensores a las unidades correspondientes de temperatura.

A continuación, el siguiente bloque (**Figura 6-7**) está dividido en tres secciones, y es en donde se convierten los valores de las lecturas en las unidades correspondientes de temperatura, humedad e índice de calor. Para los sensores analógicos, se utilizaron fórmulas compartidas por la propia compañía Vegetronix. En el caso de DHT22, la propia librería se encarga de dicha conversión.

Por otro lado, también se aprovecha para separar la parte entera y decimal de los valores de temperatura, para luego dividir la parte entera en bytes individuales, gracias a las variables creadas previamente en la función "union".

```
  // Cálculo del checksum con CRC32
  uint8_t byteBuffer[] = { ttbout.ttb[1], ttbout.ttb[0], soilTempDecimal, soilHum,
                           tdbout.tdb[1], tdbout.tdb[0], airTempDecimal, airHum,
                           icbout.icb[1], icbout.icb[0], indiceCalorDecimal };
  size_t numBytes = sizeof(byteBuffer) - 1;
  uint32_t  checksum = CRC32::calculate(byteBuffer, numBytes);
// Conversión de la variable checksum de "uint32_t" a cuatro "bytes"
  csbin.CSByte = checksum;
  for(int conv4=0 ; conv4 < sizeof(csbin.CSByte) ; conv4++)
      csbout.csb[conv4] = csbin.csb[conv4];
```

Figura 6-8. Procedimiento de cálculo del checksum con todos los datos de los sensores agrupados en una matriz.

Tras almacenar todas las lecturas y hacer las conversiones correspondientes, el siguiente paso en el bucle, mostrado en la **Figura 6-8**, es el de calcular el checksum de los datos. Para ello, se creó una matriz en la que se recopilan todas las variables de un solo byte de

tamaño, para entonces llamar a la función *CRC32::calculate*, incluida en la librería CRC32. Una vez creada la variable checksum, se procede a dividirla en cuatro bytes individuales.

```
// Procedimiento para la trama KiSS, designando primero las variables FEND, FESC, TFEND y TFESC
byte FEND = 0xC0;
byte FESC = 0xDB;
byte TFEND = 0xDC;
byte TFESC = 0xDD;
uint8_t DatosYChecksum[] = { ttbout.ttb[1], ttbout.ttb[0], soilTempDecimal, soilHum,
                             tdbout.tdb[1], tdbout.tdb[0], airTempDecimal, airHum,
                             icbout.icb[1], icbout.icb[0], indiceCalorDecimal,
                             csbout.csb[3], csbout.csb[2], csbout.csb[1], csbout.csb[0] };
int data_lenght = sizeof(DatosYChecksum);
```

Figura 6-9. Procedimiento de preparación para después poder crear la trama KISS.

Llegados a este punto, ya está casi todo preparado para crear la trama KISS. El bloque de código de la **Figura 6-9** es bastante sencillo. Simplemente se crean las variables del principio KISS, es decir, los bytes de inicio y final de trama (FEND) y los bytes para evitar errores en la lectura de datos (FESC, TFEND y TFESC). Se genera también una nueva matriz, ya no solamente con los datos sino con el checksum incluido.

```
// Envío de la trama KiSS
Serial.print(FEND, HEX);
Serial.print(" ");
  for (int b = 0; b < data_lenght; b++) {
    if (DatosYChecksum[b] == FEND) {
      Serial.print(FESC, HEX);
      Serial.print(" ");
      Serial.print(TFEND, HEX);
      Serial.print(" ");
```

Figura 6-10. Porción del procedimiento para la creación de la trama KISS, byte por byte.

Para terminar el bucle de la función "void loop", se hace un procedimiento en el que se envían los datos y el checksum por el puerto serie. En la **Figura 6-10** se puede observar el inicio de dicho procedimiento, en el que se añade el byte FEND al inicio de la trama KISS. Después, lee toda la matriz de datos y la envía por el puerto serie, byte por byte. En caso de que un byte coincida con FEND o FESC, se hace una sustitución con la finalidad de evitar errores de lectura por parte del receptor de la trama KISS.

En definitiva, lo que se obtiene al final con este código es una trama KISS que incluye un byte de inicio y final de trama (FEND), los datos obtenidos de los sensores y el checksum, todo ello en valores hexadecimales.

6.4. Códigos para la calibración de los sensores analógicos

En el **Anexo 2** y **3** se muestran los códigos para la calibración del sensor THERM200 y VH400, respectivamente. Los códigos tienen una estructura más simple en comparación a la de la prueba de concepto. Nos centramos sólo en la toma de medidas del sensor de referencia SHT10 y de los sensores analógicos THERM200 y VH400, por separado. Como sólo se quiere calibrar y no tratar de usar el menor número de bytes posible, no se ha hecho ningún bloque de "union". Además, se ha obviado el checksum y tampoco se incluye el procedimiento para crear la trama KISS.

Diferencias aparte, el resto de ambos códigos es similar, pues comienzan con la declaración de variables; se establece el "void setup"; se sigue con la toma de 150 medidas de voltaje para luego hacer una media de todos los valores; se convierten dichos valores a temperatura o humedad; y finalizan con el envío de los resultados al puerto serie.

La calibración de los sensores se ha realizado con el material disponible en condiciones de confinamiento. El procedimiento ha sido el siguiente:

- Se ha usado un vaso de agua fría, una maceta con tierra húmeda y otra con tierra seca para simular diferentes condiciones. También se hicieron medidas de la temperatura del aire en una habitación cerrada.
- Se dejó que los sensores tomaran medidas durante 45 minutos. Pasado ese tiempo, se apuntaron las últimas cuatro medidas y se hizo una media.
- Medidas en mano, se estudió la diferencia entre los valores medidos por el sensor de referencia y los sensores analógicos para terminar de ajustar la relación entre voltaje y temperatura o humedad.

La **Tabla 6-1** muestra los resultados de la calibración del sensor THERM200:

Tabla 6-1. Resultados de la calibración del sensor THERM200 con el sensor SHT10 como referencia.

Medida	Temperatura media THERM200 (ºC)	Temperatura media SHT10 (ºC)	Factor	Diferencia (ºC)
1	1,1	6,9	6,3	5,8
2	12,4	18,0	1,5	5,6
3	16,0	23,4	1,5	7,4
4	18,3	25,8	1,4	7,5

La medida 1 se hizo con los sensores metidos en un vaso de agua fría; la 2, en una maceta con tierra húmeda; la 3, en una maceta con tierra seca; y la 4 es una medida de la temperatura del aire.

No se ha podido hacer un procedimiento de calibración satisfactorio con el sensor VH400. Las medidas en la maceta con tierra húmeda eran demasiado dispares e incoherentes, pues el sensor VH400 enviaba lecturas en torno al 20 % mientras que el sensor SHT10 enviaba medidas superiores al 90 % e incluso más de 100 %. Por esa razón, no se ha incluido una tabla de resultados para esta parte de la calibración.

7. Discusión

7.1. Discusión de resultados de la prueba de concepto

La **Figura 7-1** muestra un ejemplo de cuatro tramas KISS enviadas por puerto serie, visualizadas en el monitor serial de Arduino IDE. El monitor serial es una ventana dentro del programa que nos permite ver todo lo que se envía por puerto serie.

```
C0 0 13 9 0 0 14 9 2F 0 14 3 A8 2B A6 81 C0

C0 0 11 7 0 0 15 3 2C 0 14 6 C3 CC EB B8 C0

C0 0 F 1 2 0 15 5 18 0 14 3 F0 37 78 65 C0

C0 0 10 8 1 0 15 5 17 0 14 3 A6 B5 B3 4E C0
```

Figura 7-1. Ejemplo de envío de la trama KISS con la última versión del código para la prueba de concepto.

Lo primero que salta a la vista son los bytes en los que sólo se muestra un carácter, en vez de los dos caracteres alfanuméricos que siempre se muestran en los valores hexadecimales de la trama KISS. Tras una serie de pruebas, se comprobó que esto sucede cuando el primer carácter del valor hexadecimal es un cero. Esto no implica que desaparezca; el valor hexadecimal se envía en su totalidad por puerto serie, sin eliminar ninguno de sus caracteres.

Otra verificación que se llevó a cabo fue con respecto a la sustitución de bytes con las variables FEND, FESC, TFEND y TFESC. Tomando como base el mismo procedimiento para la generación de la trama KISS, se creó otro código para comprobar que la sustitución se llevara a cabo con éxito. El resultado fue satisfactorio. El **Anexo 4** muestra el código para las pruebas relacionadas con la trama KISS.

Las cuatro tramas KISS que se muestran en la **Figura 7-1** tienen un tamaño de 17 bytes. Este valor se encuentra por debajo del tamaño máximo de datos que un nanosatélite de Aistech Space puede transmitir (192 bytes). Esto significa que, siempre y cuando no suceda ninguna sustitución de variables, se pueden enviar hasta 11 tramas KISS por

mensaje. El tamaño total de 11 tramas sería de 187 bytes como mínimo, por lo que habría 5 bytes de margen para las sustituciones.

Para demostrar que lo que hay en las tramas KISS se corresponden con las medidas de los sensores, se creó una extensión del código de lectura y envío de datos. Lo que se hizo en esta versión del código fue enviar por puerto serie las variables encargadas de almacenar las medidas en cada bloque. De esta manera, se puede comparar y comprobar que una trama KISS almacena exitosamente las medidas correctas.

La **Figura 7-2** se puede ver un ejemplo de un resultado, utilizando medidas reales con los sensores THERM200, VH400 y DHT22.

```
THERM200
Valor voltaje: 1.43V   Valor convertido: 19.52°C
Valor entero: 19   Valor decimal: 5
Valor entero almacenado con Union: 019
VH400
Valor voltaje: 1.25V   Valor convertido: 13%
DHT22
Valor de temperatura: 20.09°C   Valor entero: 20   Valor decimal: 0
Valor entero almacenado con Union: 020
Valor de humedad: 23%
Valor de índice de calor: 18.76°C   Valor entero: 18   Valor decimal: 7
Valor entero almacenado con Union: 018
CHECKSUM
Valor numérico: 541101209   Valor hexadecimal: 20408C99
Valor almacenado con Union: 3264140153
TRAMA KISS
C0 0 13 5 D 0 14 0 17 0 12 7 20 40 8C 99 C0
```

Figura 7-2. Ejemplo de envío final con todos los datos y valores intermedios, el checksum y la trama KISS.

Si tomamos el sensor THERM200 como referencia, observamos que el valor convertido a partir de la lectura de voltaje se divide de manera correcta en la parte entera y decimal. Partimos con una medida de temperatura de 19,52 ºC. El valor entero es una variable de dos bytes y da como resultado 19. El valor decimal se almacena en una variable de un byte y sólo muestra la cifra 5, pues en el código truncamos el valor para sólo almacenar el primer decimal. Seguidamente, la función "union" logra dividir el valor entero de la temperatura en dos variables de un byte de tamaño. En este caso, el primer byte guarda el valor 0, mientras que el segundo byte almacena el valor 19.

También se puede comprobar que la trama KISS envía los datos correctos, si comparamos los valores hexadecimales enviados por puerto serie con los datos obtenidos por los sensores y el checksum. Por ejemplo:

- El primer C0 corresponde al byte de apertura en la trama KISS, FEND.
- El 00 (visualizado simplemente como 0 en la imagen) corresponde al primer byte del valor entero almacenado con Union, para la medición de temperatura del sensor THERM200.
- El 13 es el segundo byte del mismo valor entero. El valor hexadecimal 13 correspondiente al número 19 en valor decimal.
- El 05 (visualizado simplemente como 5 en la imagen) corresponde al byte que almacena el valor decimal de la medición de temperatura del sensor THERM200.
- Si miramos el valor hexadecimal obtenido del checksum, corresponde byte por byte a los cuatro últimos valores hexadecimales de la trama KISS, previo al byte de término C0.

Una vez hechas las comprobaciones, hay algunos detalles a mejorar de cara a tener un código más completo, como ofrecer metadatos. Por un lado, se puede añadir una marca temporal para saber en qué hora del día se tomaron las medidas. Esto implicaría añadir un reloj absoluto en el Arduino Uno, además de ampliar el tamaño de las tramas KISS. En caso de que el terminal ya se encargue de añadir una marca temporal, no sería necesario implementarlo en el código. Por el otro, sería interesante añadir un identificador para saber a qué cultivo hacen referencia las tramas KISS.

7.2. Discusión de resultados de la calibración

A simple vista, en los resultados de la calibración se aprecia que el sensor SHT10 transmite a Arduino Uno valores superiores a los del sensor analógico THERM200.

En relación a las medidas, hay varios detalles que valen la pena discutir. Primeramente, las medidas del 2 al 4 presentan un factor entre las temperaturas medias de los sensores SHT10 y THERM200 bastante similares, pero difieren mucho con la primera medida de todas. A falta de más pruebas con ambos sensores, sería interesante disponer de más medidas con temperaturas cercanas a las del 2 al 4 y superiores, para saber si el factor se mantiene entre 1,4 y 1,5 o no.

La última columna de la **Tabla 6-1** muestra la diferencia de valores entre las temperaturas medias de cada sensor. Se observa que la diferencia va en aumento cuando sube la temperatura. Estos datos de diferencias, junto con los del factor, nos dan una idea de cuáles son las curvas de temperatura que tiene cada sensor en base al voltaje aplicado en ellos y las ecuaciones utilizadas tanto en el código como en la librería para el sensor SHT10. Como primer ajuste de calibración para el sensor THERM200, se puede considerar que podemos aplicar un factor de multiplicación de 1,5 para el cálculo de la variable "valorTherm", de manera que quede así en la línea correspondiente del código:

$$valorTherm = (41.67 * voltajeTherm - 40.00) * 1.5$$

El **Anexo 5** muestra el código completo con este ajuste.

Por falta de tiempo e imposibilidad de acceder a mejores sensores de referencia, no se ha podido terminar la calibración de los sensores analógicos. Convendría buscar un sensor de humedad con el que calibrar el sensor VH400 pues, debido a las medidas tan dispares, no fue posible con el SHT10. Con los materiales adecuados, se seguiría el mismo procedimiento que con el sensor THERM200.

Aunque no se busca exactitud, pues el enfoque del proyecto es la transmisión de datos, sería interesante comprobar con un potenciómetro la relación entre la temperatura o humedad y la lectura de tensión de los sensores analógicos. De esta manera, se podría comprobar si las ecuaciones proporcionadas por la compañía Vegetronix son correctas, pues son las que se usaron en los códigos.

8. Conclusiones

Tras un proceso de investigación sobre distintos cultivos, se han considerado el tomate y la soja como dos cultivos de interés para su monitorización mediante nanosatélites. La producción total de cada año y la cantidad de tierras dedicadas al cultivo de estos productos, tanto en el hemisferio norte como en el sur, son dos de los motivos por los que los nanosatélites pueden prestar un buen servicio de monitorización de cultivos a nivel internacional.

Para el desarrollo de la prueba de concepto de monitorización de estos cultivos y elección de sensores, se han escogido la temperatura y la humedad del aire y del suelo como parámetros a monitorizar. Con esto en mente, se optó por usar los sensores analógicos THERM200 y VH400, más el sensor digital DHT22. Junto con el microcontrolador Arduino Uno, conforman la parte principal del hardware del prototipo preliminar.

Se han conectado los tres sensores al microcontrolador, de modo que comparten la misma toma de tierra y referencia de voltaje. Para unificar las conexiones, se ha usado una placa de prototipos. Para ajustar las medidas de voltaje de los sensores analógicos, también se ha colocado una resistencia.

Se ha logrado crear un código que recopile las medidas de los tres sensores. Asimismo, el código crea el checksum de dichos datos y, posteriormente, crea con éxito una trama KISS. De cara al futuro, una mejora a implementar en el código es poner una marca temporal en cada trama KISS, para conocer en qué momento del día se ha hecho cada medida. También se propone añadir un identificador para saber a qué cultivo pertenecen las medidas.

Se ha llevado a cabo con éxito un primer calibrado del sensor analógico THERM200. Las primeras medidas nos dan una idea sobre cómo ajustar los cálculos de las lecturas de dicho sensor. Sin embargo, no fue posible hacer un calibrado con el sensor VH400.

En definitiva, se han cumplido casi todos los objetivos del proyecto, a falta de enviar las medidas a un nanosatélite para terminar de validar la prueba de concepto y de hallar un sensor de humedad para usarlo de referencia en el calibrado del sensor VH400. En lo

referente a la transmisión de datos, aunque no se ha podido realizar, se ha dejado el sistema preparado para probar la transmisión cuando se tenga acceso a los nanosatélites.

Se ha comprobado que el tamaño de los datos es apto para enviarse por nanosatélite. Siempre y cuando no haya más de 5 sustituciones de variables mediante el protocolo KISS, se pueden enviar hasta 11 tramas KISS por mensaje. Tras comparar los valores de las medidas con las variables recopiladas en cada trama KISS, se ha confirmado que los datos que hay en cada trama corresponden a las medidas de los tres sensores.

Conviene proseguir con el calibrado de los sensores, con tal de ajustar las curvas de calibración de cada sensor del prototipo y asegurar así que las medidas sean correctas. Por un lado, se debe buscar sensores cuya resolución y precisión sean más altas que las de los sensores analógicos. De esta manera, se tendrían mejores referencias para el calibrado de THERM200 y VH400, con sensores de prestaciones más altas. Por el otro, es preciso llevar a cabo más pruebas a diferentes temperaturas y humedad, de manera que abarque todo el rango representativo de la monitorización de cultivos.

Por último, debido a que la monitorización de los cultivos va más allá de la temperatura y humedad, otro paso a dar sería expandir el prototipo y utilizar sensores que sirvan para monitorizar otros parámetros como el pH o la salinidad del suelo. Además, sería interesante tomar medidas en un campo de cultivo, para simular con mayor exactitud cómo sería el proceso de tomar las medidas y enviar los datos a los nanosatélites.

9. Referencias

Adafruit Industries (2019a). *DHT22 Temperature-Humidity Sensor + Extras*. https://www.adafruit.com/product/385 [Consulta 17 de octubre de 2019]

Adafruit Industries (2019b). *DHT sensor library*. https://github.com/adafruit/DHT-sensor-library [Consulta 8 de noviembre de 2019]

Adams, S. R., Cockshull, K. E., & Cave, C. R. J. (2001). Efect of Temperature on the Growth and Development of Tomato Fruits. https://doi.org/10.1006/anbo.2001.1524

Andrades, M., & Martínez, E. (2014). Fertilidad del suelo y parámetros que la definen. 3ª ed.

Baker, C. (2017). *CRC32*. https://github.com/bakercp/CRC32 [Consulta 22 de noviembre de 2019]

Bayle, J. (2013). *C Programming for Arduino*. Packt Publishing.

Buchen, E. (2014). SpaceWorks' 2014 Nano/Microsatellite Market Assessment. En *28th Annual AIAA/USU Conference on Small Satellites*.

Chepponis, M., Karn, P. (1987). *The KISS TNC: A simple Host-to-TNC communications protocol*. http://www.ka9q.net/papers/kiss.html [Consulta 18 de diciembre de 2019]

DFRobot (2020). *Digital Temperature and Humidity sensor With Stainless Steel Probe SKU SEN0148*. https://wiki.dfrobot.com/Digital_Temperature_and_Humidity_sensor__With_Stainless_Steel_Probe___SKU_SEN0148_ [Consulta 7 de enero de 2020]

Food and Agriculture Organization of the United Nations (2019). *FAOSTAT*. http://www.fao.org/faostat/es/#data/QC [Consulta septiembre de 2019]

Ghazali, M. F., Wikantika, K., Harto, A. B., & Kondoh, A. (2019). Generating soil salinity, soil moisture, soil pH from satellite imagery and its analysis. *Information Processing in Agriculture*. https://doi.org/10.1016/j.inpa.2019.08.003

Gordon, R., & Bootsma, A. (1993). Analyses of growing degree-days for agriculture in Atlantic Canada. *Climate Research*, *3*(January 1993), 169–176. https://doi.org/10.3354/cr003169

Kalra, A., Chechi, R., & Khanna, R. (2002). Role of Zigbee Technology in Agriculture Sector, (January), 19–21.

Mcgrath, C., Wright, D., Mallarino, A. P., & Lenssen, A. W. (2013). Soybean Nutrient Needs. *Agriculture and Environment Extension Publications*, *189*.

Oxer, J., Ribble, M. (2011). *SHT1x*. https://github.com/practicalarduino/SHT1x [Consulta 7 de enero de 2020]

Piper, E. L., & Boote, K. J. (1999). Temperature and Cultivar Effects on Soybean Seed Oil and Protein Concentrations, *76*(10). https://doi.org/10.1007/s11746-999-0099-y

Ramane, D. V, Patil, S. S., & Shaligram, A. D. (2018). Detection of NPK nutrients of soil using Fiber Optic Sensor, (April).

Roldán, J. J., Cerro, J. del, Ramos, D. G., Aunon, P. G., Garzón, M., León, J. de, & Barrientos, A. (2017). Chapter 4 - Robots in Agriculture: State of Art and Practical Experiences. In *Service Robots* (pp. 67–90).

Sensirion (2020). *Digital Humidity Sensor SHT1x (RH/T)*. https://www.sensirion.com/en/environmental-sensors/humidity-sensors/digital-humidity-sensors-for-accurate-measurements/ [Consulta 7 de enero de 2020]

Starke Ayres (2014). *Tomato Production Guideline*. https://www.starkeayres.co.za/com_variety_docs/Tomato-Production-Guideline-2014.pdf [Consulta: 19 de enero de 2020]

Tridge (2019). *Market Intelligence*. https://www.tridge.com/overview [Consulta septiembre de 2019]

Vegetronix (2019a). *Soil Temperature Sensor Probes*. https://www.vegetronix.com/Products/THERM200/ [Consulta 23 de octubre de 2019]

Vegetronix (2019b). VH400 Low-Cost Soil Moisture Sensors. https://www.vegetronix.com/Products/VH400/ [Consulta 23 de octubre de 2019]

Wang, X., & Gao, H. (2016). Agriculture Wireless Temperature and Humidity Sensor Network Based on ZigBee Technology. https://doi.org/10.1007/978-3-642-27281-3_20

Anexo 1: código de lectura y envío de datos de los tres sensores

```
// Inclusión de librerías
#include "DHT.h"
#include <CRC32.h>

// Declaración de variables para el sensor DHT22
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
float rawValueDHTTemp;
float rawValueDHTHum;
int airTempEntero;
byte airTempDecimal;
byte airHum;
int indiceCalorEntero;
byte indiceCalorDecimal;

// Declaración de variables para el sensor THERM200
const int thermPin = A1;
float rawValueTherm;
int soilTempEntero;
byte soilTempDecimal;

// Declaración de variables para el sensor VH400
const int humPin = A2;
float rawValueHum;
byte soilHum;

// Función "union" para convertir variables float e int a bytes individuales
union TempThermEnteroPasoABytes{
  int TThermByte;
  byte ttb[2];
} ttbin, ttbout;

union TempDHTEnteroPasoABytes{
  int TDHTByte;
  byte tdb[2];
} tdbin, tdbout;

union IndiceCalorEnteroPasoABytes{
  int ICByte;
  byte icb[2];
} icbin, icbout;

union ChecksumPasoABytes{
  uint32_t CSByte;
```

```
  byte csb[4];
} csbin, csbout;

// Función "void setup"
void setup() {
  Serial.begin(9600);
  analogReference(EXTERNAL);
  dht.begin();
}

// Inicio de la función "void loop"
void loop() {
  rawValueTherm = 0;
  rawValueHum = 0;
  rawValueDHTTemp = 0;
  rawValueDHTHum = 0;

// Recogida de lecturas de cada sensor
  for (int i = 0; i < 150; i++) {
    rawValueTherm += analogRead(thermPin);
    rawValueHum += analogRead(humPin);
    rawValueDHTTemp += dht.readTemperature();
    rawValueDHTHum += dht.readHumidity();
    delay(43);
  }

// Conversión de los valores de las lecturas del sensor THERM200
  float voltajeTherm = (rawValueTherm / 150.0)*(3.0 / 1023.0);
  float valorTherm = 41.67*voltajeTherm-40.0;
  soilTempEntero = int(valorTherm);
  soilTempDecimal = byte((valorTherm-soilTempEntero)*10);

  ttbin.TThermByte = soilTempEntero;
  for(int conv1=0 ; conv1 < sizeof(ttbin.TThermByte) ; conv1++)
      ttbout.ttb[conv1] = ttbin.ttb[conv1];

// Conversión de los valores de las lecturas del sensor VH400
  float voltajeHum = (rawValueHum / 150.0)*(3.0 / 1023.0);
  if(voltajeHum <= 1.1) {
    soilHum = 10*voltajeHum-1;
  } else if(voltajeHum > 1.1 && voltajeHum <= 1.3) {
    soilHum = 25*voltajeHum-17.5;
  } else if(voltajeHum > 1.3 && voltajeHum <= 1.82) {
    soilHum = 48.08*voltajeHum-47.5;
  } else if(voltajeHum > 1.82) {
    soilHum = 26.32*voltajeHum-7.89;
}

// Conversión de los valores de las lecturas del sensor DHT22
  float temporalTemp = rawValueDHTTemp / 150.0;
```

```
  float temporalHum = rawValueDHTHum / 150.0;
  float indiceCalor = dht.computeHeatIndex(temporalTemp, temporalHum, false);
  airTempEntero = int(temporalTemp);
  airTempDecimal = byte((temporalTemp-airTempEntero)*10);
  airHum = temporalHum;
  indiceCalorEntero = int(indiceCalor);
  indiceCalorDecimal = byte((indiceCalor-indiceCalorEntero)*10);

  tdbin.TDHTByte = airTempEntero;
  for(int conv2=0 ; conv2 < sizeof(tdbin.TDHTByte) ; conv2++)
     tdbout.tdb[conv2] = tdbin.tdb[conv2];

  icbin.ICByte = indiceCalorEntero;
  for(int conv3=0 ; conv3 < sizeof(icbin.ICByte) ; conv3++)
     icbout.icb[conv3] = icbin.icb[conv3];

// Cálculo del checksum
  uint8_t byteBuffer[] = { ttbout.ttb[1], ttbout.ttb[0], soilTempDecimal, soilHum,
                           tdbout.tdb[1], tdbout.tdb[0], airTempDecimal, airHum,
                           icbout.icb[1], icbout.icb[0], indiceCalorDecimal };
  size_t numBytes = sizeof(byteBuffer) - 1;
  uint32_t  checksum = CRC32::calculate(byteBuffer, numBytes);

  csbin.CSByte = checksum;
  for(int conv4=0 ; conv4 < sizeof(csbin.CSByte) ; conv4++)
     csbout.csb[conv4] = csbin.csb[conv4];

// Preparación para la creación de la trama KISS
  byte FEND = 0xC0;
  byte FESC = 0xDB;
  byte TFEND = 0xDC;
  byte TFESC = 0xDD;
  uint8_t DatosYChecksum[] = { ttbout.ttb[1], ttbout.ttb[0], soilTempDecimal, soilHum,
                               tdbout.tdb[1], tdbout.tdb[0], airTempDecimal, airHum,
                               icbout.icb[1], icbout.icb[0], indiceCalorDecimal,
                               csbout.csb[3], csbout.csb[2], csbout.csb[1], csbout.csb[0] };
  int data_lenght = sizeof(DatosYChecksum);

// Creación y envío de la trama KISS por el monitor serial
Serial.print(FEND, HEX);
Serial.print(" ");
  for (int b = 0; b < data_lenght; b++) {
    if (DatosYChecksum[b] == FEND) {
      Serial.print(FESC, HEX);
      Serial.print(" ");
      Serial.print(TFEND, HEX);
      Serial.print(" ");
    }
    else if (DatosYChecksum[b] == FESC) {
      Serial.print(FESC, HEX);
```

```
      Serial.print(" ");
      Serial.print(TFESC, HEX);
      Serial.print(" ");
    }
    else {Serial.print(DatosYChecksum[b], HEX);
    Serial.print(" ");
    }
  }
      Serial.print(FEND, HEX);
  Serial.println();
  Serial.println();
}
```

Anexo 2: código final para el calibrado del sensor analógico THERM200

```
// Inclusión de librería
#include <SHT1x.h>

// Declaración de variables para el sensor SHT10
#define dataPin  10
#define clockPin 11
SHT1x sht1x(dataPin, clockPin);
float temp_c;

// Declaración de variables para el sensor THERM200
int thermPin = A1;
float soilTemp;
float rawValueTherm;

// Función "void setup"
void setup()
{
   Serial.begin(9600);
   analogReference(EXTERNAL);
   Serial.println("Starting up");
}

// Inicio de la función "void loop"
void loop()
{ temp_c = 0;
  rawValueTherm = 0;

// Recogida de lecturas de cada sensor
  for (int i = 0; i < 150; i++) {
  temp_c += sht1x.readTemperatureC();
  rawValueTherm += analogRead(thermPin);
  delay(43);
  }

// Conversión de los valores de las lecturas del sensor SHT10
  temp_c = temp_c / 150.0;

// Conversión de los valores de las lecturas del sensor THERM200
  float sensorVoltageTherm = (rawValueTherm / 150.0)*(3.0 / 1023.0);
  soilTemp = 41.67*sensorVoltageTherm-40.0;

// Envío de los resultados por el monitor serial
  Serial.print("Temperatura SHT-10: ");
  Serial.print(temp_c, DEC);
  Serial.println("ºC");
```

```
  Serial.print("Temperatura THERM200: ");
  Serial.print(soilTemp);
  Serial.println("ºC");
  Serial.println();
}
```

Anexo 3: código final para el calibrado del sensor analógico VH400

```
// Inclusión de librería
#include <SHT1x.h>

// Declaración de variables para el sensor SHT10
#define dataPin  10
#define clockPin 11
SHT1x sht1x(dataPin, clockPin);
float humidity;

// Declaración de variables para el sensor THERM200
int humPin = A1;
float rawValueHum;
float soilHum;

// Función "void setup"
void setup()
{
   Serial.begin(9600);
   analogReference(EXTERNAL);
   Serial.println("Starting up");
}

// Inicio de la función "void loop"
void loop()
{ humidity = 0;
  rawValueHum = 0;

// Recogida de lecturas de cada sensor
  for (int i = 0; i < 150; i++) {
  humidity += sht1x.readHumidity();
  rawValueHum += analogRead(humPin);
  delay(43);
  }

// Conversión de los valores de las lecturas del sensor SHT10
  humidity = humidity / 150.0;

// Conversión de los valores de las lecturas del sensor VH400
  float sensorVoltage = (rawValueHum / 150.0)*(3.0 / 1023.0);
  if(sensorVoltage <= 1.1) {
    soilHum = 10*sensorVoltage-1;
  } else if(sensorVoltage > 1.1 && sensorVoltage <= 1.3) {
    soilHum = 25*sensorVoltage-17.5;
  } else if(sensorVoltage > 1.3 && sensorVoltage <= 1.82) {
    soilHum = 48.08*sensorVoltage-47.5;
```

```
  } else if(sensorVoltage > 1.82) {
    soilHum = 26.32*sensorVoltage-7.89;
  }

// Envío de los resultados por el monitor serial
  Serial.print("Humedad SHT-10: ");
  Serial.print(humidity);
  Serial.println("%");
  Serial.print("Humedad VH400: ");
  Serial.print(soilHum);
  Serial.println("%");
  Serial.println();
}
```

Anexo 4: código de pruebas relacionadas con la trama KISS

```
// Declaración de variables
  byte FEND = 0xC0;
  byte FESC = 0xDB;
  byte TFEND = 0xDC;
  byte TFESC = 0xDD;
  uint8_t DatosYChecksum[] = { 0x4F, 0xC0, 0x07, 0x80, 0xDD,
                                       0x12, 0xBD, 0xBE, 0xAA, 0xDB};
  int data_lenght = sizeof(DatosYChecksum);

// Procedimiento de la creación de la trama KISS
void setup() {
  Serial.begin(9600);
Serial.print(FEND, HEX);
Serial.print(" ");
  for (int b = 0; b < data_lenght; b++) {
    if (DatosYChecksum[b] == FEND) {
      Serial.print(FESC, HEX);
      Serial.print(" ");
      Serial.print(TFEND, HEX);
      Serial.print(" ");
    }
    else if (DatosYChecksum[b] == FESC) {
      Serial.print(FESC, HEX);
      Serial.print(" ");
      Serial.print(TFESC, HEX);
      Serial.print(" ");
    }
    else {Serial.print(DatosYChecksum[b], HEX);
    Serial.print(" ");
    }
  }
      Serial.print(FEND, HEX);
  Serial.println();
  Serial.println();
}

void loop() {
}
```

Resultado de la trama KISS: C0 4F DB DC 7 80 DD 12 BD BE AA DB DD C0

Anexo 5: código final de lectura y envío de datos, ajustado según el resultado del calibrado

```
// Inclusión de librerías
#include "DHT.h"
#include <CRC32.h>

// Declaración de variables para el sensor DHT22
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
float rawValueDHTTemp;
float rawValueDHTHum;
int airTempEntero;
byte airTempDecimal;
byte airHum;
int indiceCalorEntero;
byte indiceCalorDecimal;

// Declaración de variables para el sensor THERM200
const int thermPin = A1;
float rawValueTherm;
int soilTempEntero;
byte soilTempDecimal;

// Declaración de variables para el sensor VH400
const int humPin = A2;
float rawValueHum;
byte soilHum;

// Función "union" para convertir variables float e int a bytes individuales
union TempThermEnteroPasoABytes{
  int TThermByte;
  byte ttb[2];
} ttbin, ttbout;

union TempDHTEnteroPasoABytes{
  int TDHTByte;
  byte tdb[2];
} tdbin, tdbout;

union IndiceCalorEnteroPasoABytes{
  int ICByte;
  byte icb[2];
} icbin, icbout;

union ChecksumPasoABytes{
  uint32_t CSByte;
```

```
  byte csb[4];
} csbin, csbout;

// Función "void setup"
void setup() {
  Serial.begin(9600);
  analogReference(EXTERNAL);
  dht.begin();
}

// Inicio de la función "void loop"
void loop() {
  rawValueTherm = 0;
  rawValueHum = 0;
  rawValueDHTTemp = 0;
  rawValueDHTHum = 0;

// Recogida de lecturas de cada sensor
  for (int i = 0; i < 150; i++) {
    rawValueTherm += analogRead(thermPin);
    rawValueHum += analogRead(humPin);
    rawValueDHTTemp += dht.readTemperature();
    rawValueDHTHum += dht.readHumidity();
    delay(43);
  }

// Conversión de los valores de las lecturas del sensor THERM200, con el ajuste
correspondiente según el resultado de la calibración
  float voltajeTherm = (rawValueTherm / 150.0)*(3.0 / 1023.0);
  float valorTherm = (41.67*voltajeTherm-40.0)*1.5;
  soilTempEntero = int(valorTherm);
  soilTempDecimal = byte((valorTherm-soilTempEntero)*10);

  ttbin.TThermByte = soilTempEntero;
  for(int conv1=0 ; conv1 < sizeof(ttbin.TThermByte) ; conv1++)
     ttbout.ttb[conv1] = ttbin.ttb[conv1];

// Conversión de los valores de las lecturas del sensor VH400
  float voltajeHum = (rawValueHum / 150.0)*(3.0 / 1023.0);
  if(voltajeHum <= 1.1) {
    soilHum = 10*voltajeHum-1;
  } else if(voltajeHum > 1.1 && voltajeHum <= 1.3) {
    soilHum = 25*voltajeHum-17.5;
  } else if(voltajeHum > 1.3 && voltajeHum <= 1.82) {
    soilHum = 48.08*voltajeHum-47.5;
  } else if(voltajeHum > 1.82) {
    soilHum = 26.32*voltajeHum-7.89;
}

// Conversión de los valores de las lecturas del sensor DHT22
```

```
  float temporalTemp = rawValueDHTTemp / 150.0;
  float temporalHum = rawValueDHTHum / 150.0;
  float indiceCalor = dht.computeHeatIndex(temporalTemp, temporalHum, false);
  airTempEntero = int(temporalTemp);
  airTempDecimal = byte((temporalTemp-airTempEntero)*10);
  airHum = temporalHum;
  indiceCalorEntero = int(indiceCalor);
  indiceCalorDecimal = byte((indiceCalor-indiceCalorEntero)*10);

  tdbin.TDHTByte = airTempEntero;
  for(int conv2=0 ; conv2 < sizeof(tdbin.TDHTByte) ; conv2++)
     tdbout.tdb[conv2] = tdbin.tdb[conv2];

  icbin.ICByte = indiceCalorEntero;
  for(int conv3=0 ; conv3 < sizeof(icbin.ICByte) ; conv3++)
     icbout.icb[conv3] = icbin.icb[conv3];

// Cálculo del checksum
  uint8_t byteBuffer[] = { ttbout.ttb[1], ttbout.ttb[0], soilTempDecimal, soilHum,
                           tdbout.tdb[1], tdbout.tdb[0], airTempDecimal, airHum,
                           icbout.icb[1], icbout.icb[0], indiceCalorDecimal };
  size_t numBytes = sizeof(byteBuffer) - 1;
  uint32_t  checksum = CRC32::calculate(byteBuffer, numBytes);

  csbin.CSByte = checksum;
  for(int conv4=0 ; conv4 < sizeof(csbin.CSByte) ; conv4++)
     csbout.csb[conv4] = csbin.csb[conv4];

// Preparación para la creación de la trama KISS
  byte FEND = 0xC0;
  byte FESC = 0xDB;
  byte TFEND = 0xDC;
  byte TFESC = 0xDD;
  uint8_t DatosYChecksum[] = { ttbout.ttb[1], ttbout.ttb[0], soilTempDecimal, soilHum,
                               tdbout.tdb[1], tdbout.tdb[0], airTempDecimal, airHum,
                               icbout.icb[1], icbout.icb[0], indiceCalorDecimal,
                               csbout.csb[3], csbout.csb[2], csbout.csb[1], csbout.csb[0] };
  int data_lenght = sizeof(DatosYChecksum);

// Creación y envío de la trama KISS por el monitor serial
Serial.print(FEND, HEX);
Serial.print(" ");
  for (int b = 0; b < data_lenght; b++) {
    if (DatosYChecksum[b] == FEND) {
      Serial.print(FESC, HEX);
      Serial.print(" ");
      Serial.print(TFEND, HEX);
      Serial.print(" ");
    }
    else if (DatosYChecksum[b] == FESC) {
```

```
      Serial.print(FESC, HEX);
      Serial.print(" ");
      Serial.print(TFESC, HEX);
      Serial.print(" ");
    }
    else {Serial.print(DatosYChecksum[b], HEX);
    Serial.print(" ");
    }
  }
     Serial.print(FEND, HEX);
  Serial.println();
  Serial.println();
}
```